BOA: The Bayesian Optimization Algorithm

Martin Pelikan, David E. Goldberg, and Erick Cantú-Paz
Illinois Genetic Algorithms Laboratory
Department of General Engineering
University of Illinois at Urbana-Champaign
{pelikan,deg,cantupaz}@illigal.ge.uiuc.edu

Abstract

In this paper, an algorithm based on the concepts of genetic algorithms that uses an estimation of a probability distribution of promising solutions in order to generate new candidate solutions is proposed. To estimate the distribution, techniques for modeling multivariate data by Bayesian networks are used. The proposed algorithm identifies, reproduces and mixes building blocks up to a specified order. It is independent of the ordering of the variables in the strings representing the solutions. Moreover, prior information about the problem can be incorporated into the algorithm. However, prior information is not essential. Preliminary experiments show that the BOA outperforms the simple genetic algorithm even on decomposable functions with tight building blocks as a problem size grows.

1 INTRODUCTION

Recently, there has been a growing interest in optimization methods that explicitly model the good solutions found so far and use the constructed model to guide the further search (Baluja, 1994; Harik et al., 1997; Mühlenbein & Paaß, 1996; Mühlenbein et al., 1998; Pelikan & Mühlenbein, 1999). This line of research in stochastic optimization was strongly motivated by results achieved in the field of evolutionary computation. However, the connection between these two areas has sometimes been obscured. Moreover, the capabilities of model building have often been insufficiently powerful to solve hard optimization problems.

The purpose of this paper is to introduce an algorithm that uses techniques for estimating the joint distribution of multinomial data by Bayesian networks in order to generate new solutions. The proposed algorithm extends existing methods in order to solve more difficult classes of problems more efficiently and reliably. By covering interactions of higher order, the disruption of identified partial solutions is prevented. Prior information from various sources can be used. The combination of information from the set of good solutions and the prior information about a problem is used to estimate the distribution. Preliminary experiments with uniformly-scaled additively decomposable problems with non-overlapping building blocks indicate that the proposed algorithm is able to solve all tested problems in close to linear time with respect to the number of fitness evaluations until convergence.

In Section 2, the background needed to understand the motivation and basic principles of the discussed methods is provided. In Section 3, the Bayesian optimization algorithm (BOA) is introduced. In subsequent sections, the structure of Bayesian networks and the techniques used in the BOA to construct the network for a given data set and to use the constructed network to generate new instances are described. The results of the experiments are presented in Section 6. The conclusions are provided in Section 7.

2 BACKGROUND

Genetic algorithms (GAs) are optimization methods loosely based on the mechanics of artificial selection and genetic recombination operators. Most of the theory of genetic algorithms deals with the so-called *building blocks* (BBs) (Goldberg, 1989). By building blocks, partial solutions of a problem are meant. The genetic algorithm implicitly manipulates a large number of building blocks by mechanisms of selection and recombination. It reproduces and mixes building blocks. However, a fixed mapping from the space of solutions into the internal representation of solutions in the al-

gorithm and simple two-parent recombination operators soon showed to be insufficiently powerful even for problems that are composed of simpler partial subproblems. General, fixed, problem-independent recombination operators often break partial solutions what can sometimes lead to losing these and converging to a local optimum. Two crucial factors of the GA success—a proper growth and mixing of good building blocks—are often not achieved (Thierens, 1995). Various attempts to prevent the disruption of important building blocks have been done recently and are briefly discussed in the remainder of this section.

There are two major approaches to resolve the problem of building-block disruption. The first approach is based on manipulating the representation of solutions in the algorithm in order to make the interacting components of partial solutions less likely to be broken by recombination operators. Various reordering and mapping operators were used. However, reordering operators are often too slow and lose the race against selection, resulting in premature convergence to low-quality solutions. Reordering is not sufficiently powerful in order to ensure a proper mixing of partial solutions before these are lost. This line of research has resulted in algorithms which evolve the representation of a problem among individual solutions, e.g. the messy genetic algorithm (mGA) (**?**), the gene expression messy genetic algorithm (GEMGA) (Bandyopadhyay et al., 1998), the linkage learning genetic algorithm (LLGA) (Harik & Goldberg, 1996), or the linkage identification by non-linearity checking genetic algorithm (LINC-GA) (Munetomo & Goldberg, 1998).

A different way to cope with the disruption of partial solutions is to change the basic principle of recombination. In the second approach, instead of implicit reproduction of important building blocks and their mixing by selection and two-parent recombination operators, new solutions are generated by using the information extracted from the entire set of promising solutions.

Global information about the set of promising solutions can be used to estimate their distribution and new candidate solutions can be generated according to this estimate. A general scheme of the algorithms based on this principle is called the estimation of distribution algorithm (EDA) (Mühlenbein & Paaß, 1996). In EDAs, better solutions are selected from an initially randomly generated population of solutions like in the simple GA. The distribution of the selected set of solutions is estimated. New solutions are generated according to this estimate. The new solutions are then added into the original population, replacing some of the old ones. The process is repeated until the termination criteria are met. However, estimating the distribution is not an easy task. There is a trade off between the accuracy of the estimation and its computational cost.

The simplest way to estimate the distribution of good solutions is to consider each variable in a problem independently and generate new solutions by only preserving the proportions of the values of all variables independently of the remaining solutions. This is the basic principle of the population based incremental learning (PBIL) algorithm (Baluja, 1994), the compact genetic algorithm (cGA) (Harik et al., 1997), and the univariate marginal distribution algorithm (UMDA) (Mühlenbein & Paaß, 1996). There is theoretical evidence that the UMDA approximates the behavior of the simple GA with uniform crossover (Mühlenbein, 1997). It reproduces and mixes the building blocks of order one very efficiently. The theory of UMDA based on the techniques of quantitative genetics can be found in Mühlenbein (1997). Some analyses of PBIL can be found in Kvasnicka et al. (1996).

The PBIL, cGA, and UMDA algorithms work very well for problems with no significant interactions among variables (Mühlenbein, 1997; Harik et al., 1997; Pelikan & Mühlenbein, 1999). However, partial solutions of order more than one are disrupted and therefore these algorithms experience a great difficulty to solve problems with interactions among the variables. First attempts to solve this problem were based on covering some pairwise interactions, e.g. the incremental algorithm using the so-called dependency trees as a distribution estimate (Baluja & Davies, 1997), the population-based MIMIC algorithm using simple chain distributions (De Bonet et al., 1997), or the bivariate marginal distribution algorithm (BMDA) (Pelikan & Mühlenbein, 1999). In the algorithms based on covering pairwise interactions, the reproduction of building blocks of order one is guaranteed. Moreover, the disruption of some important building blocks of order two is prevented. Important building blocks of order two are identified using various statistical methods. Mixing of building blocks of order one and two is guaranteed assuming the independence of the remaining groups of variables.

However, covering only pairwise interactions has been shown to be insufficient to solve problems with interactions of higher order efficiently (Pelikan & Mühlenbein, 1999). Covering pairwise interactions still does not preserve higher order partial solutions. Moreover, interactions of higher order do not necessarily imply pairwise interactions that can be detected at the level

of partial solutions of order two.

In the factorized distribution algorithm (FDA) (Mühlenbein et al., 1998), a factorization of the distribution is used for generating new solutions. The distribution factorization is a conditional distribution constructed by analyzing the problem decomposition. The FDA is capable of covering the interactions of higher order and combining important partial solutions effectively. It works very well on additively decomposable problems. The theory of UMDA can be used in order to estimate the time to convergence in the FDA.

However, the FDA requires the prior information about the problem in the form of a problem decomposition and its factorization. As an input, this algorithm gets a complete or approximate information about the structure of a problem. Unfortunately, the exact distribution factorization is often not available without computationally expensive problem analysis. Moreover, the use of an approximate distribution according to the current state of information represented by the set of promising solutions can be very effective even if it is not a valid distribution factorization. However, by providing sufficient conditions for the distribution estimate that ensure a fast and reliable convergence on decomposable problems, the FDA is of great theoretical value. Moreover, for problems of which the factorization of the distribution is known, the FDA is a very powerful optimization tool.

The algorithm proposed in this paper is also capable of covering higher order interactions. It uses techniques from the field of modeling data by Bayesian networks in order to estimate the joint distribution of promising solutions. The class of distributions that are considered is identical to the class of conditional distributions used in the FDA. Therefore, the theory of the FDA can be used in order to demonstrate the power of the proposed algorithm to solve decomposable problems. However, unlike the FDA, our algorithm does not require any prior information about the problem. It discovers the structure of a problem on the fly. It identifies, reproduces and mixes building blocks up to a specified order very efficiently.

In this paper, the solutions will be represented by binary strings of fixed length. However, the described techniques can be easily extended for strings over any finite alphabet. String positions will be numbered sequentially from left to right, starting with the position 0.

3 BAYESIAN OPTIMIZATION ALGORITHM

This section introduces an algorithm that uses techniques for modeling data by Bayesian networks to estimate the joint distribution of promising solutions (strings). This estimate is used to generate new candidate solutions. The proposed algorithm is called the Bayesian optimization algorithm (BOA). The BOA covers both the UMDA as well as BMDA and extends them to cover the interactions of higher order. The order of interactions that will be taken into account can be given as input to the algorithm. The combination of prior information and the set of promising solutions is used to estimate the distribution. Prior information about the structure of a problem as well as the information represented by the set of high-quality solutions can be incorporated into the algorithm. The ratio between the prior information and the information acquired during the run used to generate new solutions can be controlled. The BOA fills the gap between the fully informed FDA and totally uninformed black-box optimization methods. Prior information is not essential.

In the BOA, the first population of strings is generated at random. From the current population, the better strings are selected. Any selection method can be used. A Bayesian network that fits the selected set of strings is constructed. Any metric as a measure for quality of networks and any search algorithm can be used to search over the networks in order to maximize the value of the used metric. New strings are generated using the joint distribution encoded by the constructed network. The new strings are added into the old population, replacing some of the old ones. The pseudo-code of the BOA follows:

The Bayesian Optimization Algorithm (BOA)

(1) set $t \leftarrow 0$
randomly generate initial population $P(0)$

(2) select a set of promising strings $S(t)$ from $P(t)$

(3) construct the network B using a chosen metric and constraints

(4) generate a set of new strings $O(t)$ according to the joint distribution encoded by B

(5) create a new population $P(t+1)$ by replacing some strings from $P(t)$ with $O(t)$
set $t \leftarrow t+1$

(6) if the termination criteria are not met, go to (2)

In the following section, Bayesian networks and the

techniques for their construction and use will be described.

4 BAYESIAN NETWORKS

Bayesian networks (Howard & Matheson, 1981; Pearl, 1988) are often used for modeling multinomial data with both discrete and continuous variables. A Bayesian network encodes the relationships between the variables contained in the modeled data. It represents the structure of a problem. Bayesian networks can be used to describe the data as well as to generate new instances of the variables with similar properties as those of given data. Each node in the network corresponds to one variable. By X_i, both the variable and the node corresponding to this variable will be denoted in this text. Each variable corresponds to one position in strings representing the solutions (X_i corresponds to the ith position in a string). The relationship between two variables is represented by an edge between the two corresponding nodes. The edges in Bayesian networks can be either directed or undirected. In this paper, only Bayesian networks represented by directed acyclic graphs will be considered. The modeled data sets will be defined within discrete domains.

Mathematically, an acyclic Bayesian network with directed edges encodes a joint probability distribution. This can be written as

$$p(X) = \prod_{i=0}^{n-1} p(X_i|\Pi_{X_i}), \tag{1}$$

where $X = (X_0, \ldots, X_{n-1})$ is a vector of variables, Π_{X_i} is the set of parents of X_i in the network (the set of nodes from which there exists an edge to X_i) and $p(X_i|\Pi_{X_i})$ is the conditional probability of X_i conditioned on the variables Π_{X_i}. This distribution can be used to generate new instances using the marginal and conditional probabilities in a modeled data set.

The following sections discuss how to learn the network structure if this is not given by the user, and how to use the network to generate new candidate solutions.

4.1 CONSTRUCTING THE NETWORK

There are two basic components of the algorithms for learning the network structure (Heckerman et al., 1994). The first one is a scoring metric and the second one is a search procedure. A scoring metric is a measure of how well the network models the data. Prior knowledge about the problem can be incorporated into the metric as well. A search procedure is used to explore the space of all possible networks in order to find the one (or a set of networks) with the value of a scoring metric as high as possible. The space of networks can be reduced by constraint operators. Commonly used constraints restrict the networks to have at most k incoming edges into each node. This number directly influences the complexity of both the network construction as well as its use for generation of new instances and the order of interactions that can be covered by the class of networks restricted in this way.

4.1.1 Bayesian Dirichlet metric

As a measure of the quality of networks, the so-called Bayesian Dirichlet (BD) metric (Heckerman et al., 1994) can be used. This metric combines the prior knowledge about the problem and the statistical data from a given data set. The BD metric for a network B given a data set D of size N, and the background information ξ, denoted by $p(D, B|\xi)$, is defined as

$$\begin{aligned} p(D,B|\xi) &= p(B|\xi) \prod_{i=0}^{n-1} \prod_{\pi_{X_i}} \frac{m'(\pi_{X_i})!}{(m'(\pi_{X_i}) + m(\pi_{X_i}))!} \cdot \\ &\quad \prod_{x_i} \frac{(m'(x_i, \pi_{X_i}) + m(x_i, \pi_{X_i}))!}{m'(x_i, \pi_{X_i})!}, \end{aligned}$$

where $p(B|\xi)$ is the prior probability of the network B, the product over π_{X_i} runs over all instances of the parents of X_i and the product over x_i runs over all instances of X_i. By $m(\pi_{X_i})$, the number of instances in D with variables Π_{X_i} (the parents of X_i) instantiated to π_{X_i} is denoted. When the set Π_{X_i} is empty, there is one instance of Π_{X_i} and the number of instances with Π_{X_i} instantiated to this instance is set to N. By $m(x_i, \pi_{X_i})$, we denote the number of instances in D that have both X_i set to x_i as well as Π_{X_i} set to π_{X_i}.

By numbers $m'(x_i, \pi_{X_i})$ and $p(B|\xi)$, prior information about the problem is incorporated into the metric. The $m'(x_i, \pi_{X_i})$ stands for prior information about the number of instances that have X_i set to x_i and the set of variables Π_{X_i} is instantiated to π_{X_i}. The prior probability $p(B|\xi)$ of the network reflects how the measured network resembles the prior network. By using a prior network, the prior information about the structure of a problem is incorporated into the metric. The prior network can be set to an empty network, when there is no such information. In our implementation, we set $p(B|\xi) = 1$ for all networks, i.e. all networks are treated equally.

The numbers $m'(x_i, \pi_{X_i})$ can be set in various ways. They can be set according to the prior information the user has about the problem. When there is no prior in-

formation, uninformative assignments can be used. In the so-called K2 metric, for instance, the $m'(x_i, \pi_{X_i})$ coefficients are all simply set to 1 (Heckerman et al., 1994). This assignment corresponds to having no prior information about the problem. In the empirical part of this paper we will use the K2 metric.

Since the factorials in Equation 2 can grow to huge numbers, usually a logarithm of the scoring metric is used. The contribution of one node to the logarithm of the metric can be computed in $O(2^k N)$ steps where k is the maximal number of incoming edges into each node in the network and N is the size of the data set (the number of instances). The computation of an increase of the logarithm of the value of the BD metric for an edge addition, edge reversal, or an edge removal, respectively, can be computed in time $O(2^k N)$ since the total sum contribution corresponding to the nodes of which the set of parents has not changed remains unchanged as well. Assuming that k is constant, we get linear time complexity of the computation of both the contribution of one node as well as the increase in the metric for an edge addition $O(N)$ with respect to the size of the data set.

4.1.2 Searching for a Good Network

In this section, the basic principles of algorithms that can be used for searching over the networks in order to maximize the value of a scoring metric are described. Only the classes of networks with restricted number of incoming edges denoted by k will be considered.

a) $k = 0$

This case is trivial. An empty network is the best one (and the only one possible).

b) $k = 1$

For $k = 1$, there exists a polynomial algorithm for the network construction (Heckerman et al., 1994). The problem can be easily reduced to a special case of the so-called maximal branching problem for which there exists a polynomial algorithm (Edmonds, 1967).

c) $k > 1$

For $k > 1$ the problem gets much more complicated. Although for $k = 1$ there exists a polynomial algorithm for finding the best network, for $k > 1$ the problem of determining the best network with respect to a given metric is NP-complete for most Bayesian and non-Bayesian metrics (Heckerman et al., 1994).

Various algorithms can be used in order to find a good network, from a total enumeration to a blind random search. Usually, due to their effectiveness in this context, simple local search based methods are used (Heckerman et al., 1994). A simple greedy algorithm, local hill-climbing, or simulated annealing can be used. Simple operations that can be performed on a network include edge additions, edge reversals, and edge removals. Each iteration, an operation that increases the network the most is applied. Only operations that keep the network acyclic and with at most k incoming edges into each of the nodes are allowed (i.e., the operations that do not violate the constraints). The algorithms can start with an empty network, the best network with one incoming edge into each node at maximum, or a randomly generated network.

In our implementation, we have used a simple greedy algorithm with only edge additions allowed. The algorithm starts with an empty network. The time complexity of this algorithm can be computed using the time complexity of a simple edge addition and the number of edges that have to be processed at most. With the BD metric, the overall time to construct the network using the described greedy algorithm is $O(k2^k n^2 N + kn^3)$. Assuming that k is constant, we get the overall time complexity $O(n^2 N + n^3)$.

4.2 GENERATING NEW INSTANCES

In this section, the generation of new instances using a network B and the marginal frequencies for few sets of variables in the modeled data set will be described. New instances are generated using the joint distribution encoded by the network (see Equation 1).

First, the conditional probabilities of each possible instance of each variable given all possible instances of its parents in a given data set are computed. The conditional probabilities are used to generate each new instance. Each iteration, the nodes whose parents are already fixed are generated using the corresponding conditional probabilities. This is repeated until the values of all variables are generated. Since the network is acyclic, it is easy to see that the algorithm is defined well.

The time complexity of generating an instance of all variables is bounded by $O(kn)$ where n is the number of variables. Assuming that k is constant, the overall time complexity is $O(n)$.

5 DECOMPOSABLE FUNCTIONS

A function is additively decomposable of a certain order if it can be written as the sum of simpler functions defined over the subsets of variables, each of cardinality less or equal than the order of the decomposition (Mühlenbein et al., 1998; Pelikan & Mühlenbein,

1999). The problems defined by this class of functions can be decomposed into smaller subproblems. However, simple GAs experience a great difficulty to solve these decomposable problems with deceptive building blocks when these are not mapped tightly onto the strings representing the solutions (Thierens, 1995).

In general, the BOA with $k \geq 0$ can cover interactions or order $k+1$. This actually does not mean that all interactions in a problem that is order-$(k+1)$ decomposable can be covered (e.g., 2D spin-glass systems (Mühlenbein et al., 1998)). There is no straightforward way to relate general decomposable problems and what are the necessary interactions to be taken into account (or, what is the order of building blocks). By introducing overlapping among the sets from the decomposition along with scaling of the contributions of each of these sets according to some function of problem size, the problem becomes very complex. Nevertheless, the class of distributions the BOA uses is very powerful the decomposable problems with either overlapping or non-overlapping building blocks or a bounded order. This has been confirmed by a number of experiments with various test functions.

6 EXPERIMENTS

The experiments were designed in order to show the behavior of the proposed algorithm only on non-overlapping decomposable problems with uniformly scaled deceptive building blocks. For all problems, the scalability of the proposed algorithm is investigated. In the following sections, the functions of unitation used in the experiments will be described and the results of the experiments will be presented.

6.1 FUNCTIONS OF UNITATION

A function of unitation is a function whose value depends only on the number of ones in a binary input string. The function values for the strings with the same number of ones are equal. Several functions of unitation can be additively composed in order to form a more complex function. Let us have a function of unitation f_k defined for strings of length k. Then, the function additively composed of l functions f_k is defined as

$$f(X) = \sum_{i=0}^{l-1} f_k(S_i), \tag{2}$$

where X is the set of n variables and S_i for $i \in \{0, \ldots, l-1\}$ are subsets of k variables from X. Sets S_i can be either overlapping or non-overlapping and they can be mapped onto a string (the inner representation of a solution) so that the variables from one set are either mapped close to each other or spread throughout the whole string. Each variable will be required to contribute to the function through some of the subfunction. A function composed in this fashion is clearly additively decomposable of the order of the subfunctions it was composed with.

A deceptive function of order 3, denoted by 3-*deceptive*, is defined as

$$f_{3deceptive}(u) = \begin{cases} 0.9 & \text{if } u = 0 \\ 0.8 & \text{if } u = 1 \\ 0 & \text{if } u = 2 \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

where u is the number of one's in an input string.

A trap function of order 5, denoted by *trap*-5, is defined as

$$f_{trap5}(u) = \begin{cases} 4-u & \text{if } u < 5 \\ 5 & \text{otherwise} \end{cases} \tag{4}$$

A bipolar deceptive function of order 6, denoted by 6-*bipolar*, is defined with the use of the 3-*deceptive* function as follows

$$f_{6bipolar}(u) = f_{3deceptive}(|3-u|) \tag{5}$$

6.2 RESULTS OF THE EXPERIMENTS

For all problems, the average number of fitness evaluations until convergence in 30 independent runs is shown. For the 3-*deceptive* and *trap*-5 functions, the population is said to have converged when the proportion of some value on each position reaches 95%. This criterion of convergence is applicable only for problems with at most one global optimum and selection schemes that do not force the algorithm to preserve the diversity in a population (e.g. niching methods). For the 6-*bipolar* function, the population is said to have converged when it contains over a half of optimal solutions. For all algorithms, the population sizes for all problem instances have been determined empirically as a minimal size so that the algorithms converge to the optimum in all of 30 independent runs. In all runs, the truncation selection with $\tau = 50\%$ has been used (the better half of the solutions is selected). Offspring replace the worse half of the old population. The crossover rate for the simple GA has been empirically determined for each problem with one problem instance. In the simple GA, the best results have been achieved with the probability of crossover 100%. The probability of flipping a single bit by mutation has been set to 1%. In the BOA, no prior information but the maximal order of interactions to be considered has been incorporated into the algorithm.

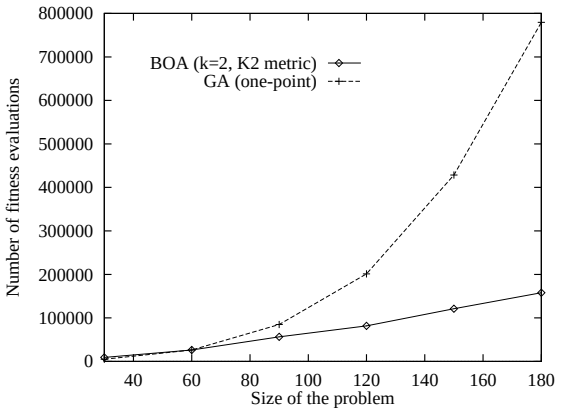

Figure 1: Results for 3-*deceptive* Function.

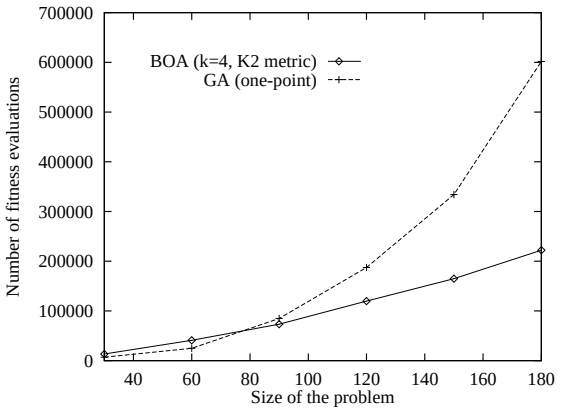

Figure 2: Results for *trap*-5 Function.

In Figure 1, the results for the 3-*deceptive* function are presented. In this function, the deceptive building blocks are of order 3. The building blocks are non-overlapping and mapped tightly onto strings. Therefore, one-point crossover is not likely to disrupt them. The looser the building blocks would be, the worse the simple GA would perform. Since the building blocks are deceptive, the computational requirements of the simple GA with uniform crossover and the BOA with $k = 0$ (i.e., the UMDA) grow exponentially and therefore we do not present the results for these algorithms. Some results for BMDA can be found in Pelikan and Mühlenbein (1999). The BOA with $k = 2$ and the K2 metric performs the best of the compared algorithms in terms of the number of functions evaluations until successful convergence. The simple GA with one-point crossover performs worse than the BOA with $k = 2$ as the problem size grows. For loose building blocks, the simple GA with one-point crossover would require the number of fitness evaluations growing exponentially

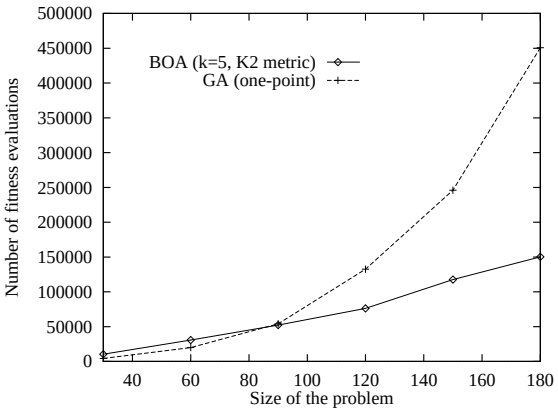

Figure 3: Results for 6-*bipolar* Function.

with the size of a problem (Thierens, 1995). On the other hand, the BOA would perform the same since it is independent of the variable ordering in a string. The population sizes for the GA ranged from $N = 400$ for $n = 30$ to $N = 7700$ for $n = 180$. The population sizes for the BOA ranged from $N = 1000$ for $n = 30$ to $N = 7700$ for $n = 180$.

In Figure 2, the results for the *trap*-5 function are presented. The building blocks are non-overlapping and they are again mapped tightly onto a string. The results for this function are similar to those for the 3-*deceptive* function. The population sizes for the GA ranged from $N = 600$ for $n = 30$ to $N = 8100$ for $n = 180$. The population sizes for the BOA ranged from $N = 1300$ for $n = 30$ to $N = 11800$ for $n = 180$.

In Figure 3, the results for the 6-*bipolar* function are presented. The results for this function are similar to those for the 3-*deceptive* function. In addition to the faster convergence, the BOA discovers a number of solutions out of totally $2^{\frac{n}{6}}$ global optima of 6-*bipolar* function instead of converging into a single solution. This effect could be further magnified by using niching methods. The population sizes for the GA ranged from $N = 360$ for $n = 30$ to $N = 4800$ for $n = 180$. The population sizes for the BOA ranged from $N = 900$ for $n = 30$ to $N = 5000$ for $n = 180$.

7 CONCLUSIONS

The experiments have shown that the proposed algorithm outperforms the simple GA even on decomposable problems with tight building blocks as the problem size grows. The gap between the proposed algorithm and the simple GA would significantly enlarge for problems with loose building blocks. For

loose mapping the time requirements of the simple GA grow exponentially with the problem size. On the other hand, the BOA is independent of the ordering of the variables in a string and therefore changing this would not affect the performance of the algorithm. The proposed algorithm works very well also for other problems with highly overlapping building blocks, e.g. spin-glasses, that are not discussed in this paper.

Acknowledgments

The authors would like to thank Heinz Mühlenbein, David Heckerman, and Ole J. Mengshoel for valuable discussions and useful comments. Martin Pelikan was supported by grants number 1/4209/97 and 1/5229/98 of the Scientific Grant Agency of Slovak Republic.

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-97-1-0050. Research funding for this project was also provided by a grant from the U.S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies and endorsements, either expressed or implied, of the Air Force of Scientific Research or the U.S. Government.

References

Baluja, S. (1994). *Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning* (Tech. Rep. No. CMU-CS-94-163). Pittsburgh, PA: Carnegie Mellon University.

Baluja, S., & Davies, S. (1997). Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. *Proceedings of the International Conference on Machine Learning*, 30–38.

Bandyopadhyay, S., Kargupta, H., & Wang, G. (1998). Revisiting the GEMGA: Scalable evolutionary optimization through linkage learning. *Proceedings of the International Conference on Evolutionary Computation (ICEC-98)*, 603–608.

De Bonet, J. S., Isbell, C. L., & Viola, P. (1997). MIMIC: Finding optima by estimating probability densities. *Advances in neural information processing systems (NIPS-97)*, *9*, 424–431.

Edmonds, J. (1967). Optimum branching. *J. Res. Nat. Bur. Standards*, *71B*, 233–240.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.

Harik, G. R., & Goldberg, D. E. (1996). Learning linkage. *Foundations of Genetic Algorithms*, *4*, 247–262.

Harik, G. R., Lobo, F. G., & Goldberg, D. E. (1997). *The compact genetic algorithm* (IlliGAL Report No. 97006). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Heckerman, D., Geiger, D., & Chickering, D. M. (1994). *Learning Bayesian networks: The combination of knowledge and statistical data* (Technical Report MSR-TR-94-09). Redmond, WA: Microsoft Research.

Howard, R. A., & Matheson, J. E. (1981). Influence diagrams. In Howard, R. A., & Matheson, J. E. (Eds.), *Readings on the principles and applications of decision analysis*, Volume II (pp. 721–762). Menlo Park, CA: Strategic Decisions Group.

Kvasnicka, V., Pelikan, M., & Pospichal, J. (1996). Hill climbing with learning (An abstraction of genetic algorithm). *Neural Network World*, *6*, 773–796.

Mühlenbein, H. (1997). The equation for response to selection and its use for prediction. *Evolutionary Computation*, *5*(3), 303–346.

Mühlenbein, H., Mahnig, T., & Rodriguez, A. O. (1998). *Schemata, distributions and graphical models in evolutionary optimization*. Submitted for publication.

Mühlenbein, H., & Paaß, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. *Parallel Problem Solving from Nature*, 178–187.

Munetomo, M., & Goldberg, D. E. (1998). *Designing a genetic algorithm using the linkage identification by nonlinearity check* (Technical Report 98014). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan Kaufmann.

Pelikan, M., & Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. *Advances in Soft Computing - Engineering Design and Manufacturing*, 521–535.

Thierens, D. (1995). *Analysis and design of genetic algorithms*. Doctoral dissertation, Katholieke Universiteit Leuven, Leuven, Belgium.